플레이리스트 기반 음악 추천 시스템

이곡어때

윤혜영 조경희 장윤식 김빈

Contents

01. 프로젝트 개요

- 주제 및 선정 배경
- 프로젝트 진행 일정
- 스택및 협업 툴
- 팀원 소개 및 역할

01-1. 주제 및 선정 배경

최근 젊은 세대들 사이에서 취향, 상황, 분위기 등에 맞는 노래를 플레이리스트 형태로 소비하는 경향이 보임

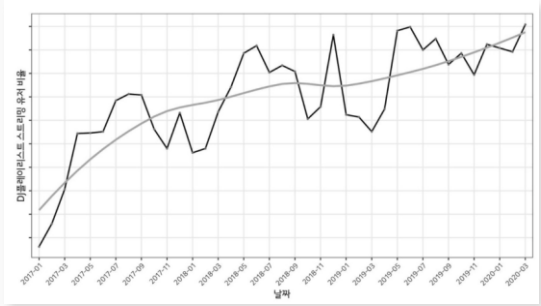
엔터테인먼트 어플 별 사용자 순위

순위	어플 명
1	YouTube
2	넷플릭스
3	멜론
4	YouTube Music
5	티빙
6	웨이브
7	지니뮤직
8	Twitch
9	아프리카TV
10	쿠팡플레이

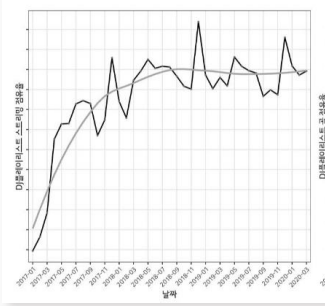
출처: 모바일 인덱스 <20대 모바일 앱 라이프 분석 리포트>
(안드로이드OS + iOS 2022년 2월 MAU 기준)

출처: 멜론 플레이리스트 데이터 – 카카오 제공 / 2017월 1월 ~ 2020년 3월 월 별 데이터

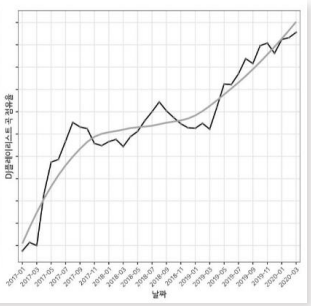
일 평균 DJ 플레이리스트 스트리밍 유저 비율



스트리밍 점유율



곡 점유율

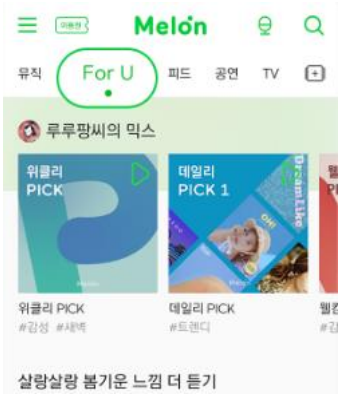


01-1. 주제 및 선정 배경

음악 스트리밍 어플 별로 다양한 방법을 도입하고 있으나, 여전히 곡 기반의 추천 시스템이 중심이 되고 있음

Melon

방대한 데이터, 카카오 추천 엔진 도입
큐레이션에 T.P.O(시간, 장소, 상황) 반영



AI 기반 개인화 큐레이션 ‘For U’
나의 이용이력 분석 ‘MY’

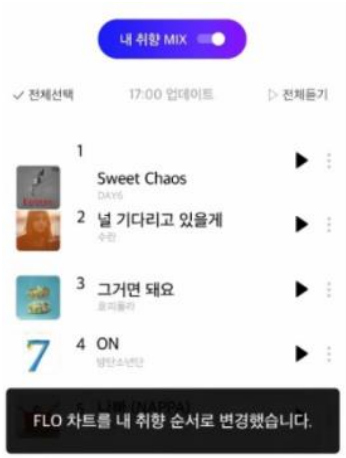
genie

컬러와 음악을 매칭한 ‘뮤직컬러’ 서비스 도입

개인 음악감상 이력 기반 ‘취향 확장형 큐레이션’
라이프스타일반영 ‘타임큐레이션’

FLO

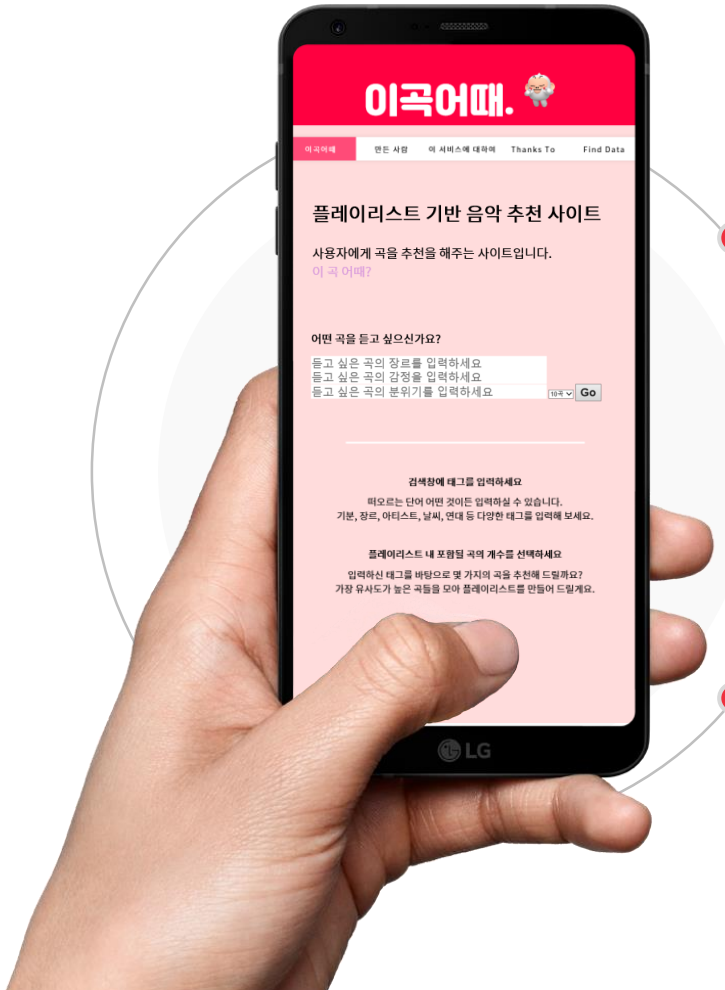
SKT T3K와 협업으로 추천 기술 고도화



취향에 따른 재생 순서 재정렬 ‘내 취향 믹스’
마음에 드는 음악과 유사한 음악 찾기 ‘점프’

01-1. 주제 및 선정 배경

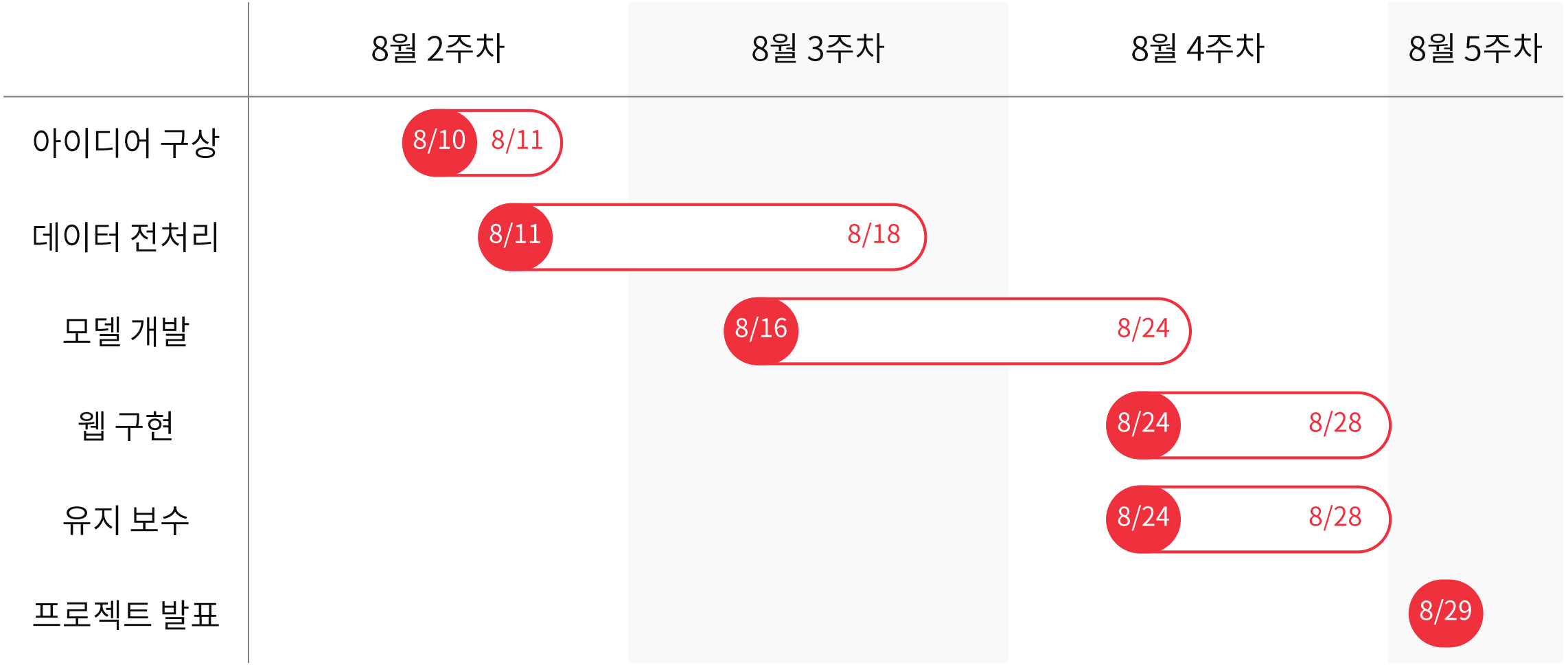
플레이리스트 기반으로 음악을 추천해주는 '이곡어때' 시스템 개발을 주제로 선정함



- 아티스트, 장르, 기분, 분위기 등 다양한 태그 입력 가능
- 붐업, 붐다운 기능을 통해 플레이리스트에 대한 만족도 평가 가능
- 태그와 유사도가 높은 곡 플레이리스트와 태그 기반으로 추출한 곡들과 유사도가 높은 곡 플레이리스트 추출
- 추출한 곡은 유튜브를 통해 바로 감상 가능

01-2. 프로젝트 진행 일정

주요 업무 진행 일정은 아래와 같음



01-3. 스택 및 협업 툴

본 프로젝트를 위해 사용한 기술 및 협업 툴은 아래와 같음

Front-End

Language


Etc.


Back-End

Collaboration Tools



01-4. 팀원 소개 및 역할

본 프로젝트는 아래 4인의 인원으로 구성됨

윤혜영

프로젝트 매니징
자연어 처리 및 모델 개발
웹 페이지 구현

조경희

전처리 및 모델 개발
웹 페이지 구현

장윤식

전처리 및 모델 개발
웹 서버 및 페이지구현

김 빈

웹 페이지 구현
모델 개발

02. 개발 과정

- 데이터 정의
- 데이터 전처리
- 모델링
- 웹 개발

02-1. 데이터 정의

카카오 아레나를 통해 다운 가능한 ‘멜론’ 데이터를 이용함

플레이리스트 정보 데이터

플레이리스트 ID

플레이리스트 제목

태그 리스트

곡 리스트

좋아요 개수

수정 날짜

(115,071개 데이터)

곡 정보 데이터

곡 ID

앨범 ID

아티스트 ID 리스트

아티스트 리스트

곡 제목

곡 장르 리스트

곡 세부 장르 리스트

발매일

(707,989개 데이터)

장르 데이터

장르 대분류

장르 소분류

02-2. 데이터 전처리

기존 데이터를 전처리한 결과는 아래와 같음

플레이리스트 정보 데이터

플레이리스트 ID

플레이리스트 제목

태그 리스트

곡 리스트

좋아요 개수

수정 날짜

(N) POS 플레이리스트명

(N) POS 태그

(N) 통합 태그

(N) 최종 태그

곡 정보 데이터

곡 ID

앨범 ID

아티스트 ID 리스트

아티스트 리스트

곡 제목

곡 장르 리스트

곡 세부 장르 리스트

발매일

(N) 곡 장르 태그

(N) 발매년도

(N) 곡 최종 태그

장르 데이터

장르 대분류

장르 소분류

02-2. 데이터 전처리 – 곡 정보 데이터

곡 정보 데이터

곡 ID

앨범 ID

아티스트 ID 리스트

아티스트 리스트

곡 제목

곡 장르 리스트

곡 세부 장르 리스트

발매일

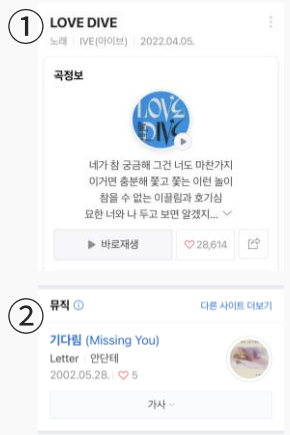
(N) 곡 장르 태그

(N) 발매년도

(N) 곡 최종 태그

발매일이 0인 데이터 2,795개 존재

네이버 노래 검색 결과 두 가지 형태에 대해 각각 크롤링 진행하여 결측치 대체



①과 ②를 비교하여 결측치 대체 값을 선정함

- ① 또는 ② 하나의 데이터만 존재할 때 – 해당 데이터 사용
- ①과 ② 모든 데이터 존재할 때 – 둘 중 더 과거의 데이터 사용

	index	artist_name_basket	song_name	search	dates_f
0	562	Bryan Adams	Summer Of `69	Bryan Adams Summer Of `69	19850107
1	785	안단테	기다림 (Missing You)	안단테 기다림 (Missing You)	20020528
2	1543	노사연	만남	노사연 만남	20150507
3	1730	Cranberries	Promises	Cranberries Promises	19990101
4	2360	Deep Purple	Hard Lovin' Woman	Deep Purple Hard Lovin' Woman	19700901

곡 장르 리스트와 세부 장르 리스트를 합쳐 장르 태그 컬럼을 생성

발매일에서 연도만 추출하여 발매년도 컬럼을 새롭게 생성

아티스트 리스트, 발매 년도, 곡 장르 태그를 통합한 후 새로운 컬럼으로 저장

02-2. 데이터 전처리 – 플레이리스트 정보 데이터

플레이리스트 정보 데이터

- 플레이리스트 ID
- **플레이리스트 제목**
- 태그 리스트
- 곡 리스트
- 좋아요 개수
- 수정 날짜
- (N) POS 플레이리스트명
- (N) POS 태그
- (N) 통합 태그
- (N) 최종 태그

플레이리스트 제목이 같은 경우 비슷한 태그와 곡을 포함하고 있으므로
중복 처리하더라도 데이터 왜곡이 크지 않을 것이라 판단하여 제목 기준으로 중복처리

정규식을 이용하여 플레이리스트 제목에서 특수문자를 제거함

```
train['plylst_title'] = train['plylst_title'].str.replace('[^가-힣A-Za-z0-9\& ]',’’)
```

string 라이브러리를 활용하여 구두점 및 특수문자를 제거하는 방법을 고려했으나
특수문자를 일일이 지정해주어야 한다는 점에서 정확도와 효율성이 떨어져 정규식을 이용

```
clean_columns = []
character_list = ['€', '£', '¥','｡ ◜   ◝ ◞｡','♪','♬;◦', '‘ಣ’, '◕ㄱ◕', '☞','☜', '花樣年華', '◇', '♡', ' └──▶']

import string
for i in columns:
  tmp = i.translate(str.maketrans('', '', string.punctuation)).strip()
  for character in character_list:
   tmp = tmp.replace(character,'')
  clean_columns.append(tmp)
```

02-2. 데이터 전처리 – 플레이리스트 정보 데이터

플레이리스트 정보 데이터

- 플레이리스트 ID
- 플레이리스트 제목
- 태그 리스트
- 곡 리스트
- 좋아요 개수
- 수정 날짜
- (N) POS 플레이리스트명
- (N) POS 태그
- (N) 통합 태그
- (N) 최종 태그

특수문자를 제거한 플레이리스트 제목에서 형태소를 어근으로 추출

```
plylst_tag = []
for i in range(len(train.plylst_title)):
 tmp = okt.pos(train.plylst_title[i], stem=True)
 plylst_tag.append(tmp)
```

Komoran, Hannanum, Okt, Kkma를 이용해 같은 문장을 형태소 분석했을 때
Okt의 결과가 가장 적합하다고 판단하여 Okt를 활용하여 형태소 분석 및 어근 추출함

```
sentence = '◯ 여행&드라이브 차안에서 함께들의면 기분좋아지는 리스트 sleep& drive◯'
```

Komoran	[('◯', 'SW'), ('여행', 'NNP'), ('&', 'SW'), ('드라이브', 'NNP'), ('차', 'NNB'), ('안', 'NNP'), ('에서', 'JKB'), ('함께', 'MAG'), ('들', 'XSN'), ('의', 'JKG'), ('면', 'NNG'), ('기분', 'NNG'), ('좋', 'VA'), ('아', 'EC'), ('지', 'VX'), ('는', 'ETM'), ('리스트', 'NNP'), ('sleep', 'SL'), ('&', 'SW'), ('drive', 'SL'), ('◯', 'SW')]
Hannanum	[('◯', 'N'), ('여행&드라이브', 'N'), ('차안', 'N'), ('에서', 'J'), ('함께들의', 'N'), ('이', 'J'), ('면', 'E'), ('기분좋', 'P'), ('아', 'E'), ('지', 'P'), ('는', 'E'), ('리스트', 'N'), ('sleep&drive◯', 'N')]
Okt	[('◯', 'Foreign'), ('여행', 'Noun'), ('&', 'Punctuation'), ('드라이브', 'Noun'), ('차안', 'Noun'), ('에서', 'Josa'), ('함께', 'Adverb'), ('들', 'Verb'), ('의', 'Noun'), ('면', 'Josa'), ('기분', 'Noun'), ('좋아지는', 'Verb'), ('리스트', 'Noun'), ('sleep', 'Alpha'), ('&', 'Punctuation'), ('drive', 'Alpha'), ('◯', 'Foreign')]
Kkma	[('◯', 'SW'), ('여행', 'NNG'), ('&', 'SW'), ('드라이브', 'NNG'), ('차안', 'NNG'), ('에서', 'JKM'), ('함께', 'MAG'), ('들', 'VV'), ('ㄹ', 'ETD'), ('의', 'NNG'), ('면', 'NNG'), ('기분', 'NNG'), ('좋아지', 'VV'), ('는', 'ETD'), ('리스트', 'NNG'), ('sleep', 'OL'), ('&', 'SW'), ('drive', 'OL'), ('◯', 'SW')]

02-2. 데이터 전처리 – 플레이리스트 정보 데이터

플레이리스트 정보 데이터

플레이리스트 ID

플레이리스트 제목

태그 리스트

곡 리스트

좋아요 개수

수정 날짜

(N) POS 플레이리스트명

(N) POS 태그

(N) 통합 태그

(N) 최종 태그

추출한 형태소 중 'Noun', 'Alpha', 'Adjective', 'Verb'을 태그로 저장

```
sentence = '◯ 여행&드라이브 차안에서 함께들의면 기분좋아지는 리스트 sleep& drive 내가 모아온 느린노래◯'
```

Okt

[('◯', 'Foreign'), ('여행', 'Noun'), ('&', 'Punctuation'), ('드라이브', 'Noun'), ('차안', 'Noun'), ('에서', 'Josa'), ('함께', 'Adverb'), ('들', 'Verb'), ('의', 'Noun'), ('면', 'Josa'), ('기분', 'Noun'), ('좋아지는', 'Verb'), ('리스트', 'Noun'), ('sleep', 'Alpha'), ('&', 'Punctuation'), ('drive', 'Alpha'), ('내', 'Noun'), ('가', 'Josa'), ('모아', 'Verb'), ('온', 'Noun'), ('느린', 'Adjective'), ('노래', 'Noun')('◯', 'Foreign')]

```
pos_plylst_tag = []

for i in plylst_tag:
  li = []
  for word, tag in i:
    if tag in ['Noun', 'Alpha', 'Adjective', 'Verb']:
      li.append(word)
  pos_plylst_tag.append(li)
```

플레이리스트명, 기존 태그 리스트에서 추출한 태그를 새로운 컬럼으로 저장

POS 태그와 POS 플레이리스트명을 통합한 후 중복 처리하여 새로운 컬럼 생성

통합 태그 컬럼을 기반으로 생성한 불용어를 제거한 태그 리스트를 컬럼으로 저장

02-2. 데이터 전처리 – 모델 학습용 데이터

모델 학습용 데이터

(index) 플레이리스트 ID

곡 ID

최종 태그

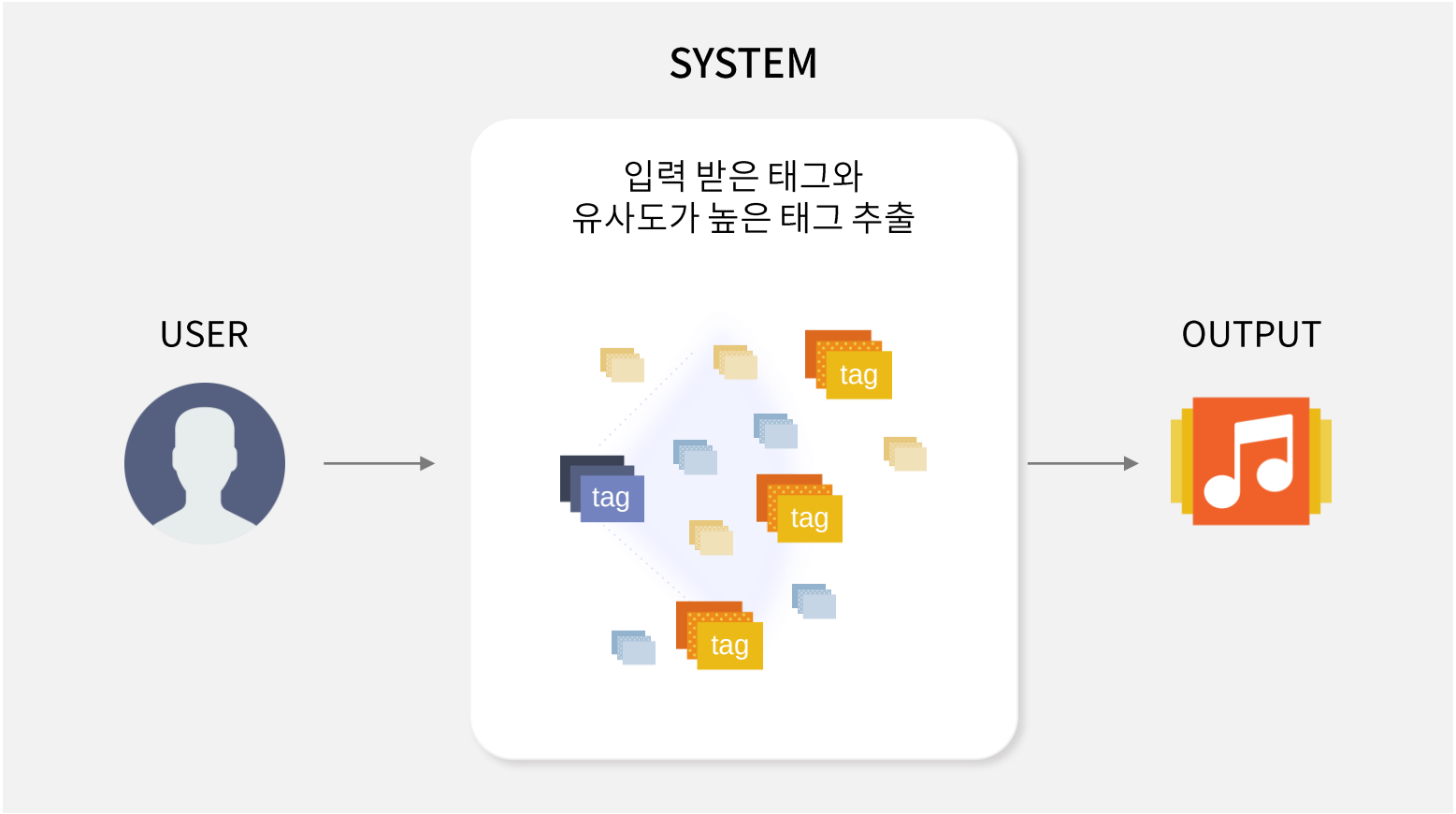
- 아티스트 이름 태그
- 장르 태그(대분류 및 소분류)
- 발행 년도
- 멜론 플레이리스트 기존 태그
- 플레이리스트 명에서 추출한 태그

	song_id	final_tags
0	0	[죽다, 회상, 아하, 2000년대, 영화음악, POP, 드라이브, 하루, 비, 추억]
1	3	[느낌, 매력, 통해, 매장, 편집, 스트레스, 슬픔, 취향, 페스티벌, 성대, 목...
2	4	[2000년대, Jude Law, 이지리스닝, newage, 뉴에이지]
3	5	[아름답다, 2000년대, 리메이크, 비, 메콩, 살, 기억, 페스티벌, 추억, 아...
4	6	[2000년대, 산책, San Francisco Symphony, 교향/관현악, 조...
...	...	...
615133	707984	[right, 90년대, 월드뮤직, nowglobal, Fela Kuti, pitc...
615134	707985	[80년대, POP, Cyndi Lauper, 로퍼, 라이벌, 마돈나, floor,...
615135	707986	[김광석, 2000년대, 매력, 들리다, 발라드, 리메이크, 업데이트, 이별, 설레...
615136	707987	[덥다, 새벽, 2000년대, 집중, 컴필레이션, 오후, 뉴에이지, Nature P...
615137	707988	['90, 록/메탈, 90년대, 샤우팅, 데뷔, 김경호, 지존]

615138 rows × 2 columns

02-3. 모델링

이곡어때 음악 추천 시스템 모델의 초기 구조는 다음과 같음

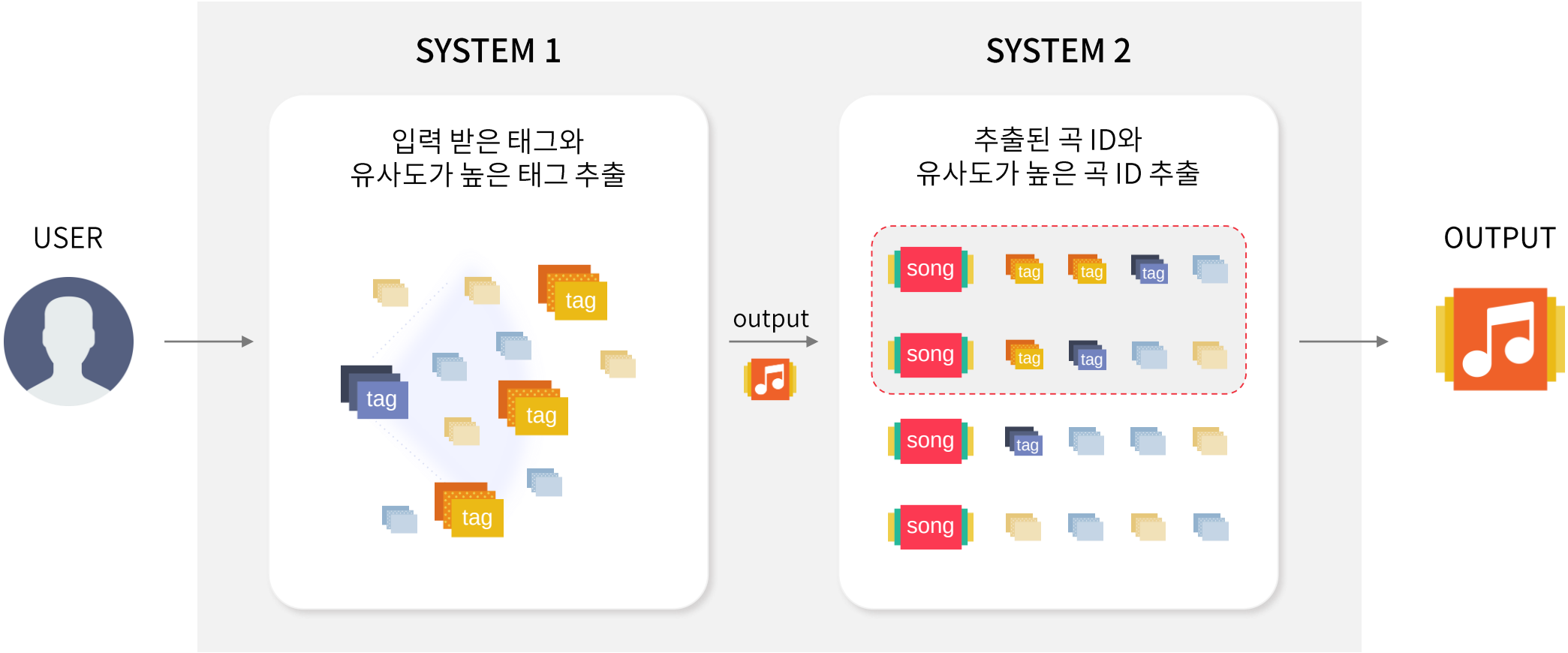


입력 받은 태그를 기반으로
유사도가 높은 태그를 추출하며,
이렇게 추출한 태그를 기반으로
플레이리스트를 생성하는 구조임

추출 태그를 기반으로
플레이리스트를 추천하는
초기 구조의 경우,
이미 데이터 내 존재하는
플레이리스트만을 결과값으로
가질 수 있다는 한계가 있었음

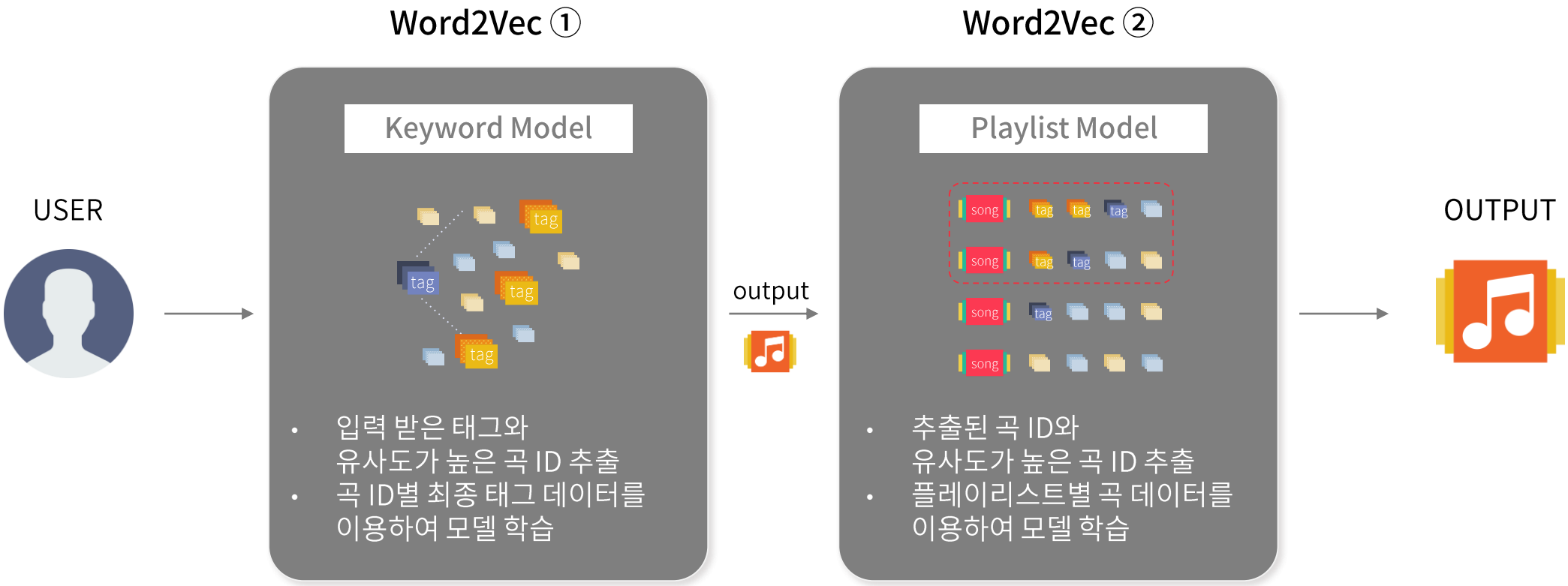
02-3. 모델링

초기 구조를 보완하여 고안한 음악 추천 시스템 모델의 기본 구조는 다음과 같음



02-3. 모델링

각 시스템별로 다른 데이터를 학습시킨 Word2Vec 모델을 결합하여 결과물을 도출함



02-3. 모델링

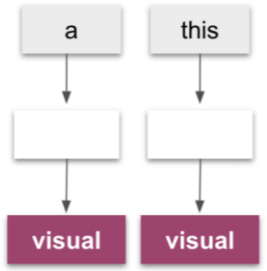
이곡어때 음악 추천 시스템 모델의 전체 구조는 아래와 같음

02-3. 모델링

Word2Vec 모델과 FastText 모델을 비교하여 결과가 더 적합한 Word2Vec 모델을 선택함

모델 학습용 데이터

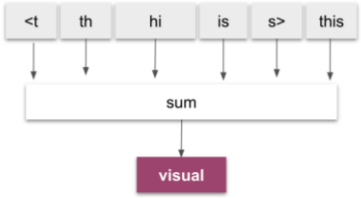
Word2Vec




```
word2vec.wv.most_similar('여름', topn=10)

[('휴양지', 0.6129440069198608),
 ('치자', 0.6072094440460205),
 ('휴가', 0.5895971059799194),
 ('바다', 0.5891450643539429),
 ('더위', 0.5835412740707397),
 ('바캉스', 0.582879364490509),
 ('땐', 0.569577693939209),
 ('청량', 0.566312313079834),
 ('해변', 0.5557003617286682),
 ('용', 0.5548463463783264)]
```

FastText




```
fasttext.wv.most_similar('여름', topn=20)

[('곧여름', 0.9917274713516235),
 ('이여름', 0.9911701083183289),
 ('낮엔여름', 0.9804202318191528),
 ('여름여름', 0.9796603322029114),
 ('이번여름', 0.9754889011383057),
 ('아직여름', 0.9749839305877686),
 ('여름여름여름', 0.9684643745422363),
 ('올여름', 0.9657432436943054),
 ('안녕여름', 0.9583413600921631),
 ('여름곡', 0.949899673461914l),
```

FastText의 경우, 입력한 단어(예: 여름)가 포함된 단어의 유사도가 높게 나타나고 있는 반면,
Word2Vec은 입력 단어가 포함되지 않더라도 의미가 유사한 경우 유사도가 높게 나타나는 것을 확인함

02-3. 모델링

Word2Vec 모델과 FastText 모델을 비교하여 결과가 더 적합한 Word2Vec 모델을 선택함

```python
def exclusion(word_list):
    count = sum([score for _, score in word_list])
    avg = count / len(word_list)

    up_word = []
    for w, s in word_list: # w : 단어, s : 유사도
        if s > avg:
            up_word.append((w, s))
    return up_word
```

모델 학습용 데이터

Word2Vec

```python
light = word2vec.wv.most_similar('새벽', topn=20)
light_word = exclusion(light)
light_word
```

```
[('밤에듣기좋은노래', 0.7161941528320312),
 ('깊은밤', 0.6749042272567749),
 ('늦은밤', 0.6706857681274414),
 ('밤공기', 0.6380821466445923),
 ('극세사', 0.6227041482925415),
 ('밤새벽', 0.6182688474655151),
 ('오지', 0.6130034923553467),
 ('불면', 0.6089522242546082),
 ('생각', 0.6083134412765503),
 ('고요한', 0.6071279644966125)]
```

FastText

```python
light = fasttext.wv.most_similar('새벽', topn=20)
light_word = exclusion(light)
light_word
```

```
[('새벽새벽', 0.9829696416854858),
 ('깊은새벽', 0.9727602005004883),
 ('짙은새벽', 0.9721930623054504),
 ('새벽비', 0.957750797271728 5),
 ('밤_새벽', 0.9570854306221008),
 ('새벽별', 0.9504075646400452),
 ('새벽증후군', 0.9489274024963379),
 ('새벽3시', 0.947988212108612 1),
 ('새벽밤', 0.947135925292968 8)]
```

평균 이상의 유사도를 갖는 태그만 가져오는 exclusion 함수를 정의한 후, 상위 20개만 확인

→ 이곡어때 음악 추천 시스템에는 Word2Vec을 채택하는 것이 적합하다고 판단하여 해당 모델을 선택함

02-3. 모델링 – Keyword Model

음악 추천 시스템 모델링을 위해 작성한 코드는 아래와 같음

```
word2vec = Word2Vec(window=3,min_count=0,workers=5,sg=1)
word2vec.build_vocab(tag_df.result_tag.values)
word2vec.train(tag_df.result_tag.values,
total_examples=len(tag_df.result_tag.values), epochs=5)

keyword_model = Word2Vec().wv.load('/content/word2vec_tag')

word = input("입력해주세요 : ")
keyword = keyword_model.wv.most_similar(word,topn=len(keyword_model.vocab))
keyword = dict(keyword)
keyword[word] = 1

test = make_score_column(keyword,tag_df)

song_id = test['song_id'].iloc[0]
```

Word2Vec 모델 학습
- 모든 빈도수의 단어를 포함하여 진행
- Skip-gram 모델을 사용한 Word2Vec 적용

입력 받은 단어를 기준으로 유사도가 높은 태그를 모두 불러와 dictionary로 생성한 후, 입력 받은 키워드의 유사도를 1로 지정

```
def make_score_new(_dict,_list):
  temp = []

  for i in _list:
     temp.append(_dict[i])
  try:
   score = sum(temp)/(len(temp))
  except:
   score = 0
  return score
```

```
def make_score_column(_dict,df):
  df['score'] = df.final_tags_chk.apply(lambda x : make_score_new(_dict,x))
  sort_df = df.sort_values('score',ascending=False)
  return sort_df
```

최종 곡-태그 데이터 각 행에 대해,
입력 받은 값과 태그 간의 유사도 평균을 구해 새로운 컬럼 [‘score’]을 생성
- 새로운 컬럼 ‘score’은 유사도가 높은 순으로 정렬되어 있음

02-3. 모델링 – Playlist Model

음악 추천 시스템 모델링을 위해 작성한 코드는 아래와 같음

```
play_list_model = Word2Vec().wv.load('/content/word2vec_playlist')
song_meta = pd.read_json('/content/song_meta.json')

similar_song = play_list_model.wv.most_similar(str(song_id), topn=20)
song_num = [(int(i[0].strip())) for i in similar_song]
song_meta.loc[song_num].song_name
```

앞서 추출한 song_id(입력어와 가장 유사도가 높은 곡)와 유사도가 높은 곡들 20개를 뽑아 출력하는 코드

>>> 입력해주세요 : 사랑

	final_tags	score
17252	[2000년대, 킨젝스, 청춘, 포크/블루스, 사랑]	0.550538
121273	[2000년대, 청춘, 포크/블루스, 사랑, 라이너스]	0.539052
248473	[포크/블루스, 사랑, 2000년대, 정지찬]	0.529451
603326	[2000년대, 해오라기, 청춘, 포크/블루스, 사랑]	0.526703
319542	[포크/블루스, 사랑, 2000년대]	0.514976

song_id(입력어와 가장 유사도가 높은 곡)

```
154856                                        어디가 끝인지
385126    Mozart : Symphony No.41 In C Major, K551 `Jupi...
650520                Blues My Naughty Sweetie Gives To Me
495439    Satie : 3 Gymnopedie No.1 - Lent Et Douloureux...
232114                          Some Heads Are Gonna Roll
221534                                      새벽을 알리는 소리
530515                                        Tenor King
611757    Bach : Goldberg Variation BWV.988 - Var.XVII (...
345693                    Creeping Death (Metallica Cover)
416989                                   Don't Look Back
651692                                     Losing Voice
232814                                   Don't Be Cruel
424842                                  당신의 화문이 되어
455364                                      Add It Up
196816                                    Turbo Killer
215854                                        My Ship
159965    Gladiator Orchestra Suite : Part 1 The Wheat /...
285842                          Dis Lui Toi Que Je T'Aime
245290                                캡틴포스 엔딩 (파워레인저 총출동)
318026    Invisible (from the Netflix Film Klaus - In Fi...
Name: song_name, dtype: object
```

02-3. 모델링

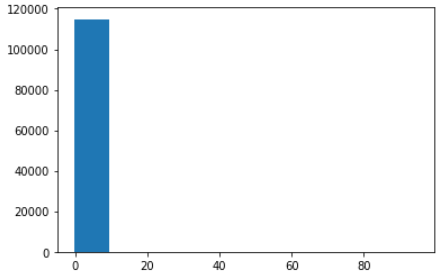
모델 최적화를 위해 score 모델을 고안하고자 함

mean normalization, min-max normalization, robust scaling 등을 시도했지만, 데이터 편차를 줄이지 못함

모델 학습용 데이터 편차 매우 큼

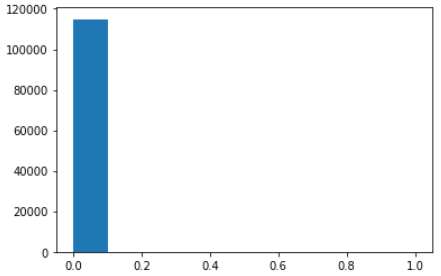
	id	like_cnt
count	115071.000000	115071.000000
mean	76728.259501	95.197687
std	44313.134619	560.653757
min	1.000000	0.000000
25%	38277.500000	2.000000
50%	76732.000000	7.000000
75%	115061.500000	24.000000
max	153428.000000	53211.000000

Mean normalization



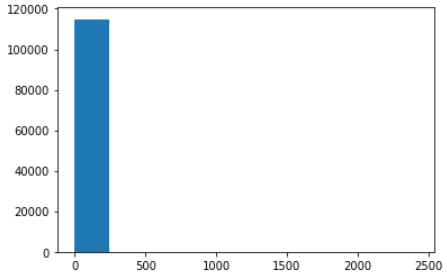
	id	norm_like_cnt
count	115071.000000	1.150710e+05
mean	76728.259501	8.181637e-18
std	44313.134619	1.000000e+00
min	1.000000	-1.697976e-01
25%	38277.500000	-1.662304e-01
50%	76732.000000	-1.573122e-01
75%	115061.500000	-1.269905e-01
max	153428.000000	9.473905e+01

Min-max normalization



	id	norm_like_cnt
count	115071.000000	115071.000000
mean	76728.259501	0.178906
std	44313.134619	1.053643
min	1.000000	0.000000
25%	38277.500000	0.003759
50%	76732.000000	0.013155
75%	115061.500000	0.045103
max	153428.000000	100.000000

Robust scaling



	id	norm_like_cnt
count	115071.000000	115071.000000
mean	76728.259501	0.001789
std	44313.134619	0.010536
min	1.000000	0.000000
25%	38277.500000	0.000038
50%	76732.000000	0.000132
75%	115061.500000	0.000451
max	153428.000000	1.000000

02-3. 모델링

모델 최적화를 위해 score 모델을 고안하고자 함

이에 플레이리스트 별 좋아요 개수, 각 곡 별 플레이리스트 내 포함 횟수의 분포를 고려하여 임의로 score model을 개발

플레이리스트 별 좋아요 개수를 기준으로 곡에 점수 부여 → 평균 점수 계산

	id	like_cnt
count	115071.000000	115071.000000
mean	76728.259501	95.197687
std	44313.134619	560.653757
min	1.000000	0.000000
25%	38277.500000	2.000000
50%	76732.000000	7.000000
75%	115061.500000	24.000000
max	153428.000000	53211.000000

점수	기준
1	25% 미만
2	25% 이상 50% 미만
3	50% 이상 75% 미만
4	75% 이상 mean값 미만
5	mean값 이상

×

각 곡 별 플레이리스트 내 포함 횟수 → 가중치 계산

```
count    615142.000000
mean          8.592928
std          39.481912
min           1.000000
25%           1.000000
50%           2.000000
75%           4.000000
max        2175.000000
Name: w, dtype: float64
```

가중치	기준
1	10회 미만
1.05	10회 이상 25% 미만
1.1	25% 이상 50% 미만
1.15	50% 이상 75% 미만
1.2	75% 이상

최종 가중치를 기준으로 데이터를 정렬했을 때, 특정 가수의 노래가 지나치게 높은 가중치를 가지는 것을 확인할 수 있었음
→ 해당 가중치를 모델에 적용하면 데이터 왜곡으로 이어질 것을 우려하여 기존 모델을 사용하는 것으로 결정함

02-4. 웹 개발

이곡어때 음악 추천 시스템을 웹으로 구현할 때 Django를 이용함

- MVT(Model-View-Template) 구조
- 파이썬을 서버측 언어로 사용할 수 있음
- 사전 생성된 파이썬 파일 구조를 이용할 수 있음
- ORM(Object-Relational Mapper)을 내장하고 있어 다양한 데이터베이스에 유연하게 액세스 가능

- 일반적인 MVC(Model-View-Controller) 구조
- 파일을 수동으로 만들어야 함
- ORM 기능을 사용하기 위해서는 SQLAlchemy라는 DB 주입 패키지 필요

Flask도 고려대상 었지만 풀스택 프레임워크로 웹 개발 전반을 경험하고 확장성을 고려하여 Django를 선택함

02-4. 웹 개발

이곡어때 음악 추천 시스템을 웹 구현할 때 고려한 기본 구조는 아래와 같음

02-4. 웹 개발

웹 구현 시 발생한 이슈는 아래 내용으로 해결함

Header와 Footer를 중복 작성하지 않기 위한 Javascript

cdn 보안 문제로 Javascript가 적용되지 않아 html만 보임

서버에 올리면 보이기 때문에 큰 문제는 아니지만
크롬 확장 프로그램인 Web Server for Chrome을 설치하면
로컬에서도 정상적으로 확인할 수 있음

CSS 및 Javascript를 수정해도 HTML에 적용되지 않는 경우 발생

웹 브라우저에서 HTML을 열면,
수정한 파일이 아닌 캐시로 저장된 내용을 불러와
HTML에 최신 수정 내용을 반영하지 못하는 경우 多

Header 안에서 css의 주소를 넣을 때
'?after'을 입력해서 최신 수정 파일을 불러오도록 함

검색 결과 페이지에서 곡 제목과 가수를 표 형식으로 함께 띄우기

1) to_html() 메소드를 이용하여
python 데이터프레임 형식 데이터를
HTML파일로 넘기는 방법을 적용함

→ 해당 메소드의 style 기본 값이 있어
css 일부가 적용되지 않는 이슈가 있었음
→ 메소드에 'border=0' 옵션으로 style 기본 값 제거

2) HTML 파일로 넘기게 되면 노래 제목에 링크를 다는 것이
불가능해 짐 → to_html을 사용하는 대신,
데이터프레임에 들어가는 데이터를 zip으로 묶어 넘겨주고
HTML 내에서 for문으로 테이블을 생성함

03. 결과

- 음악 추천 모델
- 웹 구현
- 시연 영상
- 한계점

03-1. 음악 추천 모델

이곡어때 음악 추천 시스템을 실행한 결과는 아래와 같음

사용자

태그 입력

Word2Vec

태그 유사도를 이용한
추천 점수 계산

① Output

추천1 추천2

['발라드', '슬프다', '잔잔하다']에 가장 적합한 노래들을 출력합니다.

	노래 제목	가수
1	그래..	['Seha']
2	사랑한 만큼	['김동건']
3	그대 없이는 아무것도	['멜로디슈']
4	이럴거였으면	['멜로디슈']
5	나를 보내야	['김민종']
6	비처럼 음악처럼 (Remastering)	['조덕배']
7	몰랐나요	['페이지']
8	Lovin U	['김태훈']
9	이젠 사랑희래	['주비스']
10	별이삼삼	['유정란 (유세윤 & 이정)']

유사한 곡
ID 반환

② Output

추천1 추천2

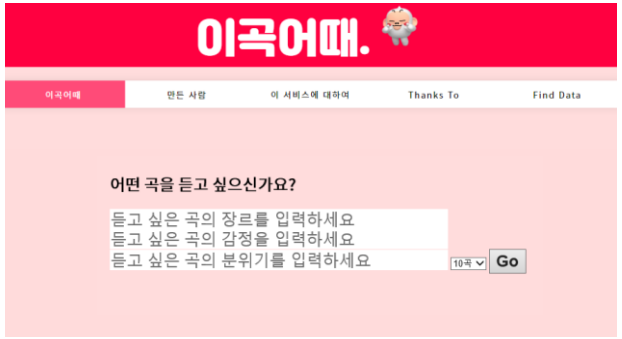
	노래 제목	가수
1	죽을만큼 보고 싶다고	['더 데이지 (The Daisy)']
2	여기까지	['전태익']
3	지워야 할 사람	['버블 시스터즈']
4	하루만 (Solo Ver.)	['손호영']
5	찻잔(노고지리)	['Various Artists']
6	그 새벽 한강에서	['김동욱']
7	왜 그런거죠	['더 데이지 (The Daisy)']
8	제자리걸음	['레아']
9	그 향기가 나서 (Feat. 태영)	['진아']
10	사랑 후	['조남욱']

03-2. 웹 구현

이곡어때 음악 추천 사이트는 아래와 같이 구현함

< 메인 페이지 >

< 검색 페이지 >



< 결과 페이지 >

< 페이지 소개 >

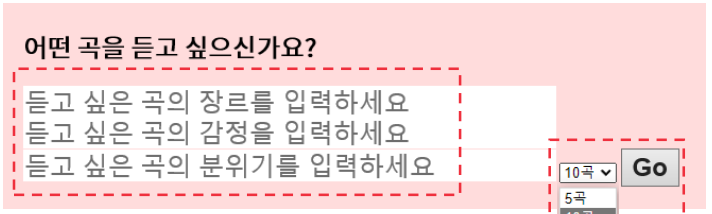
< 팀 소개 >

< Thanks To >

< Find Data>

03-2. 웹 구현 – 검색 페이지

이곡어때 음악 추천 사이트는 아래와 같이 구현함

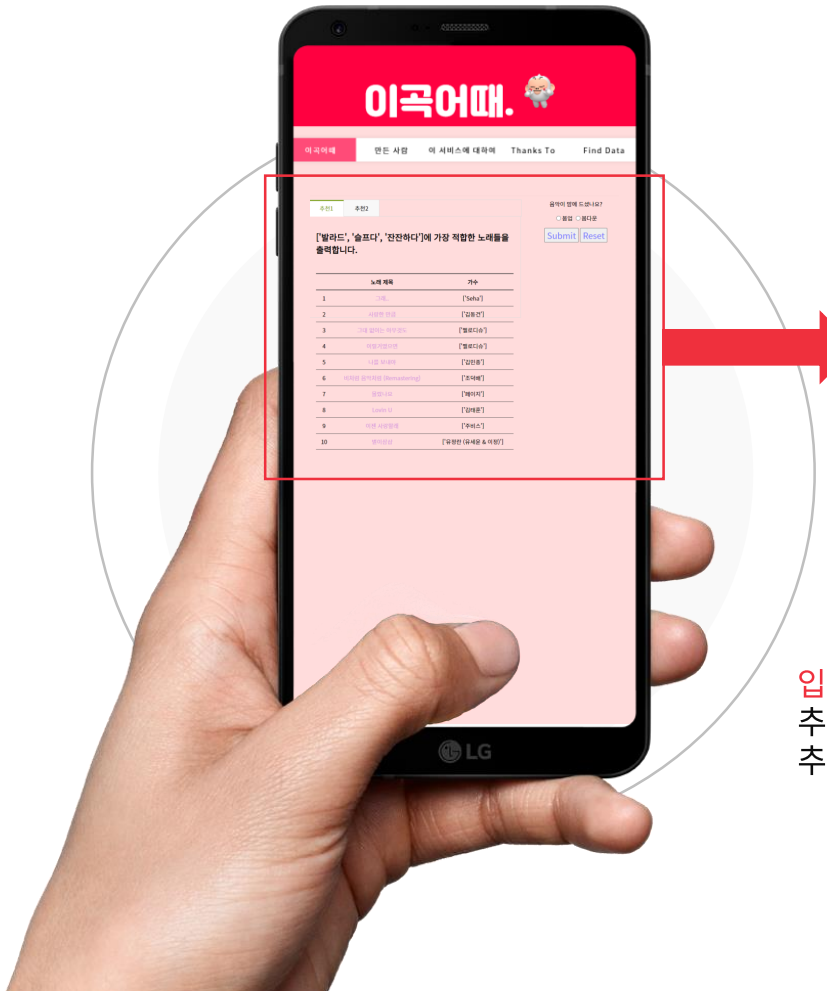


여러 개의 태그를 입력 가능
다양한 태그를 입력할수록
구체적인 결과를 도출할 수 있음

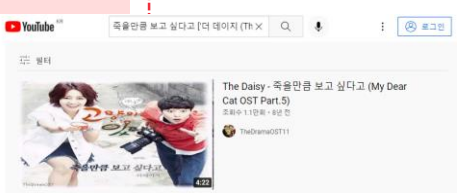
추천 받고자 하는 곡 개수 설정 가능
기본 값은 10곡이며,
유저의 선호에 따라 5곡 단위로 선택할 수 있음

03-2. 웹 구현 – 결과 페이지

이곡어때 음악 추천 사이트는 아래와 같이 구현함



추천 플레이리스트에 대한 평가 가능
붐업 또는 붐다운으로 평가하며,
평가 결과는 Find Data 페이지에서 확인할 수 있음



유튜브로 바로 듣기 가능
노래 제목과 유튜브 링크를 연결하여
검색 결과를 바로 확인할 수 있음

입력 태그에 대해 2가지 추천 결과 제공
추천 1) 입력 태그를 기반으로 가장 유사도가 높은 곡들 추천
추천 2) 입력 태그를 기반으로 추출한 최고 유사도 곡과 유사한 곡을 추천

03-3. 시연 영상


이곡어때.
이곡어때
만든 사람
이 서비스에 대하여
Thanks To
Find Data
게시판
곡 추천 어플리케이션
사용자에게 곡을 추천을 해주는 앱입니다.
이 곡 어때?

03-4. 한계점

OOV 문제를 완전히 해결하지 못함

Word2Vec 모델의 학습 방법으로 인한 OOV(Out of Vocabulary) 문제

1) 본 문제를 해결하기 위해 사전 학습된 Word2Vec 모델을 fine-tuning하는 방법을 아래와 같이 적용해 봄

1-1) 한국어 사전학습 모델인 ko.bin을 불러와 fine-tuning 후 모델 적용 시도
-> 사전학습 모델 말뭉치에 있는 단어가 결과로 반환되어 노래를 추천할 수 없었기 때문에
이곡어때 시스템을 위한 모델로 적절하지 않다고 판단함

1-2) 사전학습 모델을 fine-tuning한 후 벡터 값만 가져오고, 입력 값으로 들어올 태그들 사이의 코사인 유사도를 계산하면
그 중 유사도가 높은 태그를 관련 태그로 설정할 수 있을 것이라고 생각하여 적용 시도
-> 가능했지만 우리 데이터에 없는 단어들로 인해 태그 간 유사도에 왜곡이 생겨 사전학습 모델을 사용하지 않기로 결정함

2) 검색 페이지에서 카테고리를 제공함으로써 사용자가 입력할 수 있는 값을 제한하여 오류 발생 가능성을 줄이려는 시도를 했으나,
보다 근본적인 해결방법을 고안해 낼 필요가 있음